Scientia Iranica E (2013) 20 (3), 832–838

Sharif University of Technology

Scientia Iranica

Transactions E: Industrial Engineering

www.sciencedirect.com

Two metaheuristics for solving the reliability redundancy allocation problem to maximize mean time to failure of a series–parallel system

A.A. Najafi [a,1], H. Karimi [b], A. Chambari [c], Fatemeh Azimi [d,*]

[a] *Faculty of Industrial Engineering, K.N. Toosi University of Technology, Tehran, P.O. Box 1999143344, Iran*

[b] *Faculty of Industrial and Mechanical Engineering, Qazvin Branch, Islamic Azad University, Qazvin, P.O. Box 34185-1416, Iran*

[c] *Young Research Club, Qazvin Branch, Islamic Azad University, Qazvin, P.O. Box 34185-1416, Iran*

[d] *Department of Mathematics, Qazvin Branch, Islamic Azad University, Qazvin, P.O. Box 34185-1416, Iran*

Received 29 December 2011; revised 19 September 2012; accepted 10 October 2012

KEYWORDS

Reliability;
Optimal design;
Monte Carlo;
Genetic algorithms;
Simulated annealing.

Abstract The redundancy allocation problem is one of the main branches of reliability optimization problems. Traditionally, the redundancy allocation model has focused mainly on maximizing system reliability at a predetermined time. Hence, in this study, we develop a more realistic model, such that the mean time to failure of a system is maximized. To overcome the structural complexity of the model, the Monte Carlo simulation method is applied. Two metaheuristics, Simulated Annealing (SA) and Genetic Algorithm (GA), are proposed to solve the problem. In addition, the design of experiments and response surface methodology are employed for tuning the GA and SA parameters. The metaheuristics are compared, based on their computation time and accuracy, in 30 test problems. Finally, the results are analyzed and discussed, and some conclusions are drawn.

1. Introduction

The Redundancy Allocation Problem (RAP) is a widely investigated topic in the field of reliability optimization. The RAP involves finding a suitable allocation for the components of a system possibly under cost, weight, or other system constraints. To obtain such an allocation, many researchers have considered different configurations, such as series, parallel, *k*-out-of-*n*, and series–parallel structures. The series–parallel system structure (Figure 1) is commonly used in many system designs. A system with a series–parallel structure is composed of a fixed number of parallel subsystems connected in series. In each subsystem, redundant components can be added to improve system reliability.

Since the introduction of a mathematical model for redundancy allocation by Fyffe et al. [1], the RAP has attracted increasing attention. Fyffe et al. [1] considered the RAP with weight and cost constraints and solved it by using dynamic programming. Chern [2] showed that even a simple RAP in series systems with linear constraints is Non-deterministic Polynomial-time hard (NP-hard). This has prompted researchers to develop metaheuristic methods to achieve approximately acceptable solutions. Ida et al. [3] and Yokota et al. [4] designed a Genetic Algorithm (GA) for the RAP in a series system. Coit and Smith [5] proposed a GA to analyze series–parallel systems and to determine the optimal design configuration when there are multiple component choices available for each of several *k*-out-of-*n* subsystems. Coit and Smith [6] considered discrete component choices for each subsystem of a series–parallel system and proposed a genetic algorithm to solve the problem. Kulturel-Konak et al. [7] proposed a tabu search algorithm to solve the RAP with component mixing. Kim et al. [8] applied a simulated annealing algorithm for redundancy optimization with multiple component choices. Liang and Smith [9] proposed an ant colony optimization algorithm for solving the RAP. Nahas et al. [10] coupled ant colony and the degraded

* Corresponding author.
E-mail addresses: aanajafi@kntu.ac.ir (A.A. Najafi), hamidkzz@yahoo.com (H. Karimi), amir.chambari@gmail.com (A. Chambari), ff_azimi@yahoo.com (F. Azimi).
[1] Tel.: +98 912 3591669; fax: +98 21 88674858.

Peer review under responsibility of Sharif University of Technology.

Production and hosting by Elsevier

1026-3098 © 2013 Sharif University of Technology. Production and hosting by Elsevier B.V. Open access under CC BY-NC-ND license.
doi:10.1016/j.scient.2012.12.022

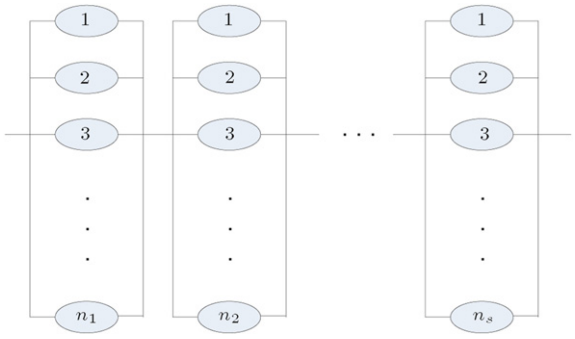

Figure 1: The structure of series–parallel system.

ceiling algorithm for the redundancy allocation problem of series–parallel systems. Liang and Chen [11] presented a variable neighborhood search algorithm for redundancy allocation of series–parallel systems. Nahas and Dao [12] applied the harmony search algorithm to the RAP, and Yeh and Hsieh [13] solved the problem using an artificial bee colony algorithm. In summary, many researchers have considered different variations of the RAP (refer to [14,15] for a survey of this problem).

In research into RAP so far, the objective has been to maximize system reliability at a predetermined time. However, it is difficult for the decision maker to determine a specific time for maximizing system reliability when the decision maker's goal is to increase the lifetime of the system. Hence, in this study, we develop a more realistic model in which the mean time to failure (MTTF) of the system is maximized. To overcome the structural complexity of the objective function, the Monte Carlo simulation method is applied. We compared two metaheuristics, GA and Simulated Annealing (SA), to solve the problem and evaluate the efficiency of these algorithms through some test problems.

The rest of the paper is organized as follows. Section 2 describes the problem, defines the assumptions, and formulates the problem. In Section 3, we propose the metaheuristics which are applied to solve the problem. Section 4 describes the tuning of parameters of the algorithms and investigates their performance. Finally, Section 5 concludes the paper.

2. Problem description

This article is about the RAP without component mixing in series–parallel systems. The redundancy strategy is cold-standby in which the redundant components do not fail before they operate. When the current component fails, one of the redundant units is switched on to continue system operation. At any time the switch is required, there is a constant probability that the switching will be successful. It is also assumed that in each subsystem, there is only one type of component to be selected and only the level of redundancy must be determined. The goal is to maximize the mean time to failure of the system considering cost and weight constraints.

2.1. Notation

s: number of subsystems;
x_i: number of components assigned to subsystem i ($i = 1, 2, \ldots, s$);
u_i: maximum x_i allowed (i.e. upper bound for x_i);
$R_i(t)$: reliability of component i at time t;
c_i: per unit cost of component i;
w_i: per unit weight of component i;
W: maximum weight of the system;
ρ: failure-detection/switching success probability;
C: maximum cost of the system;
$R_{\text{sys}}(t)$: system reliability at time t;
$MTTF$: mean time to failure of the system.

2.2. Assumptions

- The state of each component at any point of time is one of the "good" or "failed" states.
- The state of each component is independent of the other components.
- In the case of a component failure, the component is not repaired.
- Mixed component is not allowed.

2.3. Mathematical model

$$\text{Maximize MTTF} = \int_0^{\infty} R_{\text{sys}}(t)dt, \tag{1}$$

subject to the following constraints:

$$\sum_{i=1}^{s} c_i x_i \leq C, \tag{2}$$

$$\sum_{i=1}^{s} w_i x_i \leq W, \tag{3}$$

$$1 \leq x_i \leq u_i. \tag{4}$$

The objective function (1) maximizes the mean time to failure of the system. Constraints (2) and (3) represent the available cost and weight constraints, respectively. Constraint (4) denotes the domain of the decision variables.

Chern [2] showed that the RAP is NP-hard, and thus, this model is NP-hard as a generalization of the RAP. Therefore, the use of two metaheuristics to solve the RAP is justified.

3. Metaheuristics

In this section, well-known metaheuristics that have been successfully applied to combinatorial problems by researchers, such as the GA and SA, are developed to solve the RAP.

3.1. Genetic algorithm

The GA is one of the most effective and commonly known metaheuristics that have been designed to solve combinatorial problems. Each solution in GA is known as a chromosome, which is shown by a structure named solution representation. The quality of each chromosome is evaluated by a fitness function. In GA, a group of initial chromosomes is generated as the initial population. In order to create the next generation, chromosomes are selected using a selection mechanism, to which genetic operators, named crossover and mutation, are applied. This procedure continues until a stopping criterion is met. The elements of the proposed genetic algorithm are described below.

3.1.1. Solution representation

Each individual chromosome, $I = (x_1, x_2, \ldots, x_s)$, is a vector of s genes, where s denotes the number of subsystems and

Subsystem index	1	2	3	4	5	6	7	8	9	10
Number of component	5	4	3	6	3	7	4	4	2	6

Figure 2: Structure of the solution representation.

Table 1: Proposed simulation algorithm.

For $k = 1$ to *n.simulation*
  Generate a random number with the components distribution function for all components (a_{ij}) $j = 1, \ldots, s$; $i = 1, \ldots, x_j$
  For $j = 1$ to s
    $b_{j=}a_{1j}$
    For $i = 2$ to x_i
      Generate random number $r \in (0, 1)$
      If $r < \rho$
        $b_{j=}b_j + a_{ij}$
      else
        exit from for i
    End for i
  End for j
$TTF_k = \min\{b_j\}$
End for k
$MTTF = \frac{\sum_{k=1}^{n.simulation} TTF_k}{n.simulation}$

x_i denotes the number of components assigned to subsystem i ($i = 1, 2, \ldots, s$). An example of the solution representation is illustrated in Figure 2.

3.1.2. Fitness function

In order to evaluate each chromosome, we calculate the value of the fitness function by subtracting a penalty term from the objective function (1). To overcome the structural complexity of the objective function, we use Monte Carlo simulation. The Monte Carlo method is a class of computational algorithm that relies on repeated random sampling to compute the result. The basic structure of the proposed simulation algorithm is presented in Table 1, where the following notations are used:

n.simulation: number of simulations
a_{ij}: generated random number with its distribution for component i in subsystem j;
s: number of subsystems;
x_j: number of components in subsystem j;
b_j: time to failure of subsystem j in a single simulation;
ρ: failure-detection/switching success probability;
TTF_k: time to failure of the system in the kth simulation;
MTTF: mean time to failure of the system.

The number of simulations in this approach is not fixed. The simulations are performed in groups of 100, and the mean time to failure is calculated for each group. Group simulations are performed repeatedly until the means time to failure converges.

3.1.3. Penalty function

The use of crossover and mutation operators may result in an infeasible solution. This problem can be solved in different ways. Coit and Smith [16] showed that the use of a penalty function in the GA for the RAP significantly enhances the quality of the solutions. The penalty function used in this study is obtained by the following equation [16]:

$$f_{ip} = f_i - \left(\left(\frac{\Delta w_i}{NFT_w} \right)^k + \left(\frac{\Delta c_i}{NFT_c} \right)^k \right) (f_{\text{all}} - f_{\text{feas}}). \quad (5)$$

In the above equation, f_{ip} is the penalized objective function value of solution i, f_i is the unpenalized objective function value of solution i, f_{all} denotes the unpenalized value of the best solution yet to be found, and f_{feas} denotes the value of the best feasible solution yet to be found. The exponent, k, is a pre-set severity parameter. Δw_i and Δc_i are the magnitudes of any constraint violations that occur for the ith solution vector, and NFT_c and NFT_w are the near-feasible thresholds (NFT_s) of the cost and weight constraints, respectively. The dynamic *NFT* is defined as follows:

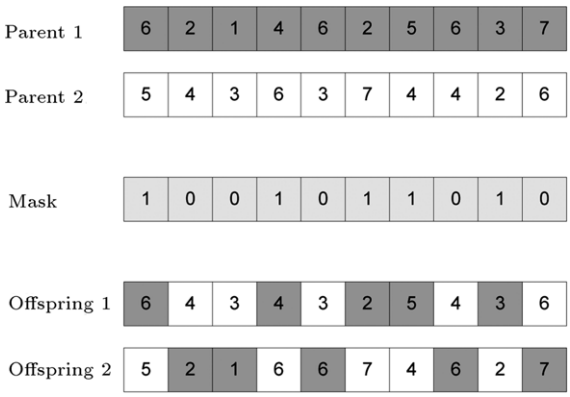

Figure 3: Uniform crossover.

$$NFT = \frac{NFT_0}{1 + \lambda g}, \quad (6)$$

where NFT_0 is an upper bound for *NFT*, g is the generation number, and λ is a constant that ensures that the entire region between NFT_0 and zero (strict feasibility) is searched.

3.1.4. Crossover

One offspring is produced by a single uniform crossover operator. The uniform crossover operator first generates a random crossover mask and then exchanges related genes between parents, according to that mask. A crossover mask is a binary string with a size the same as that of a chromosome. While scanning the mask string from left to right, if the current bit is found to be 1, then the genes at the corresponding position of the first parent are selected; otherwise, the genes at the corresponding position of the second parent are selected. Thus, one offspring is produced. Similarly, the second offspring is produced by repeating the process again, but with 0 and 1 in the mask string being exchanged. This method is illustrated in Figure 3.

If p is the population size, then p offspring are produced by crossover and are combined with the parents. Then, the p numbers of best solutions among them are selected, which survive to the next generation. The best solution within the population is never chosen for mutation, to ensure that this solution is never altered and to improve the rate of convergence of the mean lives.

3.1.5. Mutation

After the chromosome with mutation probability p_m is selected, the following procedure is adopted for all genes within the selected chromosome. A random number, r, between 0 and 1, with a continuous uniform distribution, is generated for each gene. If r is less than 0.1, then the content of the gene will increase by one, and if r is between 0.1 and 0.2, then the content of the gene will decrease by one; otherwise, the content of the gene remains unaffected. If the redundancy level of subsystem i is 1, the subsystem is only permitted to increase, and if the level is equal to u_i, it is only permitted to decrease.

3.1.6. Selection

We use a ranking-based selection method proposed by Mayerle [17], as given by the following formula:

$$select(Y) = \left\{ y_i \in Y | j = N - \left\lfloor \frac{-1 + \sqrt{1 + 4rnd(N^2 + N)}}{2} \right\rfloor \right\}, \quad (7)$$

where Y is a list ($Y = (y_1, y_2, \ldots, y_N)$) with N individuals sorted in decreasing order according to their fitness values and *rnd* is a uniformly distributed random number between 0 and 1. Eq. (7) returns the position of the individual to be selected from list Y. The formula favors the selection of individuals that are placed in initial positions in the list, that is, the best individuals are selected.

3.1.7. Stopping criteria

The algorithm terminates when the best chromosome of any given generation remains constant for 50 generations.

3.2. Simulated annealing

The SA algorithm is based on that of Metropolis et al. [18], which was originally proposed as a means of finding the equilibrium configuration of a collection of atoms at a given temperature. Pincus [19] first noted the correlation between this algorithm and mathematical minimization, but Kirkpatrick et al. [20] were the first to propose that it could form the basis of an optimization technique for combinatorial and other problems. The SA algorithm starts out with a random starting solution. A new solution is generated in the neighborhood of the current solution. If it is better than the current solution, it replaces the current solution. If the new solution is not an improvement upon the current solution, it replaces the current solution with a probability of ($P = e^{-\Delta/KT}$), where Δ is the variation in the objective function and T is a control parameter, namely, temperature. The temperature is reduced after each execution of SA to decrease the probability of accepting the non-improving solutions. This process is repeated until a stopping criterion is met.

The basic structure of the SA algorithm is presented in Table 2, where the following notations are used:

IS: initial solution;
CS: current solution;
BS: best solution;
NS: neighboring solution;
$f(X)$: objective function at solution X;
i: repetition counter;
T: temperature;
T_0: initial temperature;
K: Boltzmann constant;
L: number of repetitions allowed at each temperature level;
P: probability of acceptance *NS* when it is not better than *CS*.

The common elements of the proposed SA algorithm are described below.

3.2.1. Solution representation

The solution representations in the SA algorithm and the GA are the same.

Table 2: Simulated annealing algorithm.

```
Initialize the SA control parameter (T0, L)
Select an initial solution, IS
set T = T0; set CS = IS; set BS = IS; calculate f(IS)
while the stop criterion is not reached do:
  Set i = 1;
  while i < L do:
    generate solution NS in the neighborhood of CS; calculate
Δ = f(CS) − f(NS);
    if Δ ≤ 0
      CS = NS
    else
      generate a random number r ∈ (0, 1)
      if r ≤ (p = e^(Δ/kT));
        CS = NS;
    if f(CS) > f(BS)
      BS = CS;
    i = i + 1;
  end
  reduce the temperature T;
end
```

3.2.2. Neighbour generation

To generate a neighbouring solution in the SA algorithm, first, two random numbers are generated from the discrete uniform distribution function, where the lower limit is 1 and the upper limit equals the number of subsystems. For the corresponding cells of these random numbers, the following procedure is adopted.

A random number, r, between 0 and 1 is generated. If r is less than 0.5, then the content of the gene will increase by one; otherwise, the content will decrease by one. In the case when the solution is infeasible, the solution is omitted and the procedure is repeated.

3.2.3. Cooling scheme

After the internal loop is repeated L times, the temperature decreases according to the following formula:

$$T_i = \beta\, T_{i-1}. \quad (8)$$

β is a constant number between 0 and 1 that should be tuned. T_{i-1} is the current temperature and T_i is the new temperature.

4. Genetic algorithm and simulated annealing: fine-tuning, testing, and comparisons

In this section, the performances of the proposed algorithms have been compared. However, the values of the computational results obtained using the GA and SA are usually sensitive to the GA and SA parameters. Hence, this research first optimizes the parameter values using statistical methods. Then, the computational performance of the GA and SA on a set of test problems is assessed. Because the RAP, with the objective of maximizing the mean time to failure of the system, is a newly defined problem, no standard test problems could be obtained to assess the performance of the proposed GA and SA procedures. Therefore, random selective test problems had to be employed.

4.1. Tuning the genetic algorithm parameters

It is important to consider the method of setting the parameter values of a GA before implementing it. Modifying the GA parameters considerably affect the performance of the algorithm [21]. Conventionally, GA parameters are set on

Table 3: Search ranges and levels of the GA parameters.

Parameter	Range	Low	Middle	High
Population size	n–$3n$	n	$2n$	$3n$
α	1–5	1	3	5
λ	0.0001–0.1	0.0001	0.05	0.1
Mutation probability	0.05–0.5	0.05	0.275	0.5

Table 4: Optimum values of the GA parameters.

Parameter	Optimum value
Population size	$1.6n$
α	3
λ	0.06
Mutation probability	0.25

Table 5: Search ranges and levels of the SA parameters.

Parameter	Range	Low	Middle	High
T_0	2–18	2	10	18
L	20–140	20	80	140
α	0.95–0.99	0.95	0.975	0.99
K	0.0025–0.5	0.0025	0.25	0.5

Table 6: Optimum values of the SA parameters.

Parameter	Optimum value
T_0	18
L	20
α	0.968
K	0.0025

the basis of a trial-and-error procedure. In this section, the parameters of the proposed GA, population size, α, λ and mutation probability, are first tuned because these parameters are considered as input variables. Table 3 lists the search ranges and levels of the input variables.

Because four parameters exist, we employ a fractional factorial design with 2^{4-1} factorial points, 2×4 axial points, and four center points requiring 20 experiments. Because we want to find the GA parameter values, such that both the solution accuracy and the corresponding solution time of the algorithm are simultaneously optimized, we need to solve a bi-objective decision-making problem with conflicting objectives. In this research, the weighted additive model developed by Tiwari et al. [22] is employed to solve the bi-objective decision-making problem.

Finally, the optimum values of the GA parameters are obtained (Table 4).

4.2. Tuning the simulated annealing parameters

In this section, the parameters of the proposed SA, T_0, L, α and K, are tuned. Table 5 lists the search ranges and the levels of the input variables.

Because four factors exist, we employ a fractional factorial design with 2^{4-1} factorial points, 2×4 axial points, and four center points, requiring 20 experiments. The approach used for tuning the GA parameters is adopted for tuning the SA parameters. Table 6 shows the optimum values of the SA parameters.

4.3. Experiments and comparisons

In this section, the results obtained by evaluating the proposed fine-tuned GA and SA on some test problems are reported. The heuristic algorithms are coded and compiled in C++, and the computational experiments are performed on a PC with a Pentium 1860 processor and 1 GB RAM. The experiments are performed on the 30 test problems that we generated.

The parameters of the test problems are generated randomly. One feature of the objective function obtained using the Monte Carlo simulation is that subsystem components can have any failure distribution function, such as normal, exponential or uniform distributions.

As mentioned above, to generate the test problems, each problem for each of the algorithms was run five times.

We list the results of the experiments on and comparison between the two heuristic algorithms, GA and SA, in Table 7. For each case, the following notations are used:

n: number of problems;
Sub: number of system components;
#: number of instances when the algorithm found a solution that was better than that found by the other algorithm;
Best: best solution among those obtained from five runs of the problems for each algorithm;
RD: relative deviation from the best-known solution;
CPU: average computational time of the algorithm in seconds;
ARD: average relative deviation from the best-known solution;
MRD: maximal relative deviation from the best-known solution.

Table 7 lists the results obtained in terms of a comparison between the metaheuristics.

It should be noted that the GA performs better than SA. The GA is more efficient than SA, both in terms of the number of best solutions found, and the average and maximal relative deviations; however, the average computational time of SA is better than that of the GA.

5. Conclusions

To develop a more realistic redundancy allocation model, we developed a model such that the mean time to failure of a system was maximized. To solve the RAP, we applied two metaheuristic algorithms, GA and SA. To improve the efficiency of the proposed GA and SA, their parameters were fine-tuned using response surface methodology. These metaheuristics were compared on the basis of computational experiments performed on a set of 30 problem instances. The results of the computational experiments showed that the GA performed better than the SA algorithm, measured by solution accuracy indexes. However, the average computational time of SA was better than that of the GA. Some extensions of this research might be of interest, while, in this paper, we only considered a series–parallel system; some other configurations, such as k-out-of-n, may be considered. The other extension of this research would be to develop a bi-objective model to optimize mean time to failure and cost of the system.

Acknowledgments

The authors thank the anonymous referees for their useful suggestions that improved the presentation of the paper.

Table 7: Results of the experiments.

n	Sub	GA				SA			
		#	Best	RD (%)	CPU (s)	#	Best	RD (%)	CPU (s)
1	5	0	62.83	0.13	0.00	5	72.42	0.00	0.00
2	7	2	117.55	0.00	2.32	3	116.51	0.00	1.23
3	9	2	48.99	0.00	1.21	3	48.91	0.00	1.32
4	11	5	96.09	0.00	4.00	0	89.15	0.07	3.21
5	13	5	127.03	0.00	10.23	0	81.45	0.55	11.21
6	15	5	108.67	0.00	11.45	0	92.21	0.17	10.09
7	17	3	49.58	0.00	5.56	2	48.46	0.02	4.56
8	19	1	38.64	0.02	6.01	4	39.45	0.00	3.21
9	21	1	51.37	0.09	6.08	4	51.76	0.00	4.56
10	23	3	75.5	0.00	14.43	2	72.55	0.04	13.04
11	25	3	47.77	0.00	10.21	2	47.56	0.00	5.21
12	27	3	50.56	0.00	13.21	2	48.54	0.04	5.98
13	29	5	79.62	0.00	40.21	0	60.25	0.32	16.87
14	31	4	38.57	0.00	16.21	1	37.82	0.01	8.12
15	33	4	41.49	0.00	26.21	1	40.95	0.01	10.12
16	35	3	38.07	0.02	23.54	2	39.17	0.00	9.21
17	37	5	41.57	0.00	31.54	0	39.25	0.05	10.65
18	39	5	91.81	0.00	100.1	0	67.56	0.35	30.54
19	41	5	41.61	0.00	32.34	0	39.81	0.04	10.76
20	43	4	41.7	0.00	44.21	1	39.41	0.05	16.21
21	45	5	56.88	0.00	68.21	0	49.19	0.15	32.1
22	47	5	57.24	0.00	72.12	0	52.89	0.08	37.21
23	49	5	35.32	0.00	43.21	0	31.82	0.10	15.32
24	51	5	70.82	0.00	121.12	0	51.49	0.37	33.12
25	53	5	34.79	0.00	59.12	0	31.59	0.10	21.23
26	55	5	55.99	0.00	104.34	0	48.15	0.16	33.65
27	57	5	34.66	0.00	63.9	0	26.57	0.30	20.00
28	59	5	52.29	0.00	128.21	0	42.84	0.22	30.12
29	61	5	30.28	0.00	70.12	0	24.3	0.24	17.31
30	63	5	48.08	0.00	110.21	0	33.48	0.43	26.43
	ARD			0.008				0.129	
	MRD			0.13				0.55	

References

[1] Fyffe, D.E., Hines, W.W. and Lee, N.K. "System reliability allocation and a computational algorithm", *IEEE Transactions on Reliability*, 17, pp. 64–69 (1968).

[2] Chern, M.S. "On the computational complexity of reliability redundancy allocation in a series system", *Operations Research Letters*, 11, pp. 309–315 (1992).

[3] Ida, K., Gen, M. and Yokota, T. "System reliability optimization with several failure modes by genetic algorithm", *16th International Conference on Computers and Industrial Engineering*, Ashikaga, Japan, pp. 349–352 (1994).

[4] Yokota, T., Gen, M. and Ida, K. "System reliability of optimization problems with several failure modes by genetic algorithm", *Japanese Journal of Fuzzy Theory and Systems*, 7, pp. 117–135 (1995).

[5] Coit, D.W. and Smith, A. "Reliability optimization of series–parallel systems using a genetic algorithm", *IEEE Transactions on Reliability*, 45, pp. 254–260 (1996).

[6] Coit, D.W. and Smith, A. "Considering risk profiles in design optimization for series–parallel systems". *The 1997 Annual of the Reliability and Maintainability Symposium*, Philadelphia, USA, pp. 271–277 (1997).

[7] Kulturel-Konak, S., Smith, A. and Coit, D.W. "Efficiently solving the redundancy allocation problem using tabu search", *IIE Transactions*, 35, pp. 515–526 (2003).

[8] Kim, H., Bae, C. and Park, S. "Simulated annealing algorithm for redundancy optimization with multiple component choices", *The 2004 Asian International Workshop on Advanced Reliability Modelling*, Hiroshima, Japan, pp. 237–244 (2004).

[9] Liang, Y.C. and Smith, A. "An ant colony optimization algorithm for the redundancy allocation problem", *IEEE Transactions on Reliability*, 53, pp. 417–423 (2004).

[10] Nahas, N., Nourelfath, M. and Ait-Kadi, D. "Coupling ant colony and the degraded ceiling algorithm for the redundancy allocation problem of series–parallel systems", *Reliability Engineering and System Safety*, 92, pp. 211–222 (2007).

[11] Liang, Y.-C. and Chen, Y.-C. "Redundancy allocation of series–parallel systems using variable neighborhood search algorithms", *Reliability Engineering and System Safety*, 92, pp. 323–331 (2007).

[12] Nahas, N. and Dao, T.-M. "Harmony search algorithm: application to the redundancy optimization problem", *Engineering Optimization*, 42(9), pp. 845–861 (2010).

[13] Yeh, W.-C. and Hsieh, T.-J. "Solving reliability redundancy allocation problems using an artificial bee colony algorithm", *Computers and Operations Research*, 38(11), pp. 1465–1473 (2011).

[14] Kuo, W. and Prasad, V.R. "An annotated overview of system-reliability optimization", *IEEE Transactions on Reliability*, 49, pp. 176–187 (2000).

[15] Chambari, A., Rahmati, S.H.A., Najafi, A.A. and Karimi, A. "A bi-objective model to optimize reliability and cost of system with a choice of redundancy strategies", *Computers and Industrial Engineering*, 63, pp. 109–119 (2012).

[16] Coit, D.W. and Smith, A. "Penalty guided genetic search for reliability design optimization", *Computers and Industrial Engineering*, 30(4), pp. 895–904 (1996).

[17] Mayerle, S.F. "A genetic algorithm for the traveling salesman problem", Technical Report, Santa Catarina, Brazil (1996).

[18] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H. and Teller, E. "Equations of state calculations by fast computing machines", *Journal of Chemical Physics*, 21, pp. 1087–1092 (1958).

[19] Pincus, M. "A Monte Carlo method for the approximate solution of certain types of constrained optimization problems", *Operations Research*, 18, pp. 1225–1228 (1970).

[20] Kirkpatrick, S., Gerlatt, C.D. and Vecchi, M.P. "Optimization by simulated annealing", *Science*, 220, pp. 671–680 (1983).

[21] Shahsavar, M., Najafi, A.A. and Niaki, S.T.A. "Statistical design of genetic algorithms for combinatorial optimization problems", *Mathematical Problems in Engineering*, 2011, pp. 1–17 (2011).

[22] Tiwari, R.N., Dharmar, S. and Rao, J.R. "Fuzzy goal programming—an additive model", *Fuzzy Sets and Systems*, 24, pp. 27–34 (1987).

Amir Abbas Najafi received his B.S. degree in Industrial Engineering from Isfahan University of Technology, Iran, in 1996, and his M.S. and Ph.D. degrees in Industrial Engineering from Sharif University of Technology, Tehran, Iran, in 1998 and 2005, respectively. He is currently Assistant Professor of Industrial Engineering at K.N. Toosi University of Technology, Tehran, Iran. His research interests include project scheduling and management, portfolio selection models, reliability engineering and applied operations research.

Hamid Karimi received his B.S. degree in Computer Engineering from Zanjan Islamic Azad University, Iran, and his M.S. degree in Industrial Engineer-

ing from Qazvin Islamic Azad University, Iran. His research interests include: applications of operations research in redundancy allocation problems, project and production scheduling and, more specifically, modeling and solution methods including exact procedures (branch and bound and dynamic programming), simulation method and artificial intelligence techniques (meta-heuristic and knowledge-based algorithms).

Amirhossain Chambari received his B.S. degree in Industrial Engineering from Islamic Azad University, Gachsaran, Iran, in 2007, and his M.S. in Industrial Engineering from Islamic Azad University, Qazvin, Iran, in 2010. He currently works as Manager of the Faraz Sanat Aref Co., Tehran, Iran. His research interests include: reliability engineering, facility location-allocation, queuing system and artificial intelligence techniques.

Fatemeh Azimi obtained a B.S. degree in Mathematics from Tarbiat Moalem University, Iran, in 1999, and an M.S. degree in Mathematics from Islamic Azad University, Tehran, Iran, in 2001. Her research interests include: applied mathematics, operations research, and scheduling.